4

AD-A206 918

Technical Document 1482
February 1989

An Asynchronous Interface Between a Natural Language Query Interpreter and a Database Management System

L.E. Gadbois

ADMINISTRATIVE INFORMATION

The work reported here was performed by the author in the Decision Support and Artificial Intelligence Systems Branch. Research funding was provided by DARPA under program element number 62301EZ.

Released by
D. C. Eddington, Head
Decision Support and AI
Systems Branch

Under authority of
W. T. Rasmussen, Head
Command Support
Technologies Division

BP

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution is unlimited.
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S) NOSC TD 1482	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION Naval Ocean Systems Center | 6b. OFFICE SYMBOL (if applicable) Code 444	7a. NAME OF MONITORING ORGANIZATION
6c. ADDRESS (City, State and ZIP Code) San Diego, California 92152-5000	7b. ADDRESS (City, State and ZIP Code)
8a. NAME OF FUNDING/SPONSORING ORGANIZATION Defense Advanced Research Projects Agency | 8b. OFFICE SYMBOL (if applicable) DARPA	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c. ADDRESS (City, State and ZIP Code) DARPA-Naval Technology Office 1400 Wilson Blvd. Arlington, VA 22209	10. SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO.: 62301EZ PROJECT NO.: CD40 TASK NO.: DARPA AGENCY ACCESSION NO.: DN488 858

11. TITLE (include Security Classification)

AN ASYNCHRONOUS INTERFACE BETWEEN A NATURAL LANGUAGE QUERY INTERPRETER AND A DATABASE MANAGEMENT SYSTEM

12. PERSONAL AUTHOR(S)

L. E. Gadbois

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
Final	FROM Feb 1988 TO Apr 1988	February 1989	24

16. SUPPLEMENTARY NOTATION

17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	natural language interprocess communication IPC databases

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

This report documents computer software which interfaces a data base management system (DBMS) to a program which generates database requests. The software described controls the passing of Structured Query Language (SQL) commands into ORACLE and the capture of its output for return to the program which made the request. The software was designed for a DBMS running on a UNIX computer to be accessed by a program on the same or a remote computer. This remote computer can be running any hardware or operating system which has File Transfer Protocol (FTP) connection with the UNIX machine.

[illegible]

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT ☐ UNCLASSIFIED/UNLIMITED ☒ SAME AS RPT ☐ DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION UNCLASSIFIED
22a. NAME OF RESPONSIBLE INDIVIDUAL L. E. Gadbois	22b. TELEPHONE (include Area Code) (619) 553-2804 | 22c. OFFICE SYMBOL Code 522

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

DD FORM 1473, 84 JAN

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

SUMMARY

PROBLEM

A human–computer interface to a database has been developed which can accept typed English sentence questions from a human user and translate them into Structured Query Language (SQL) format database requests. To perform test and evaluation of the Natural Language (NL) interface and to provide a complete usable product, a database and software to interface the database with the NL program was needed. This report describes the structure and installation of the database, and the software interface between the database and the NL program.

OBJECTIVE

To write small, simple, robust code that would be easy to maintain, and could be used with little modification by other programs which need to access a database. High speed database access time was not a design requirement. This provided the freedom to write much more simple and generic code with a sacrifice of about one to two seconds delay per database access.

APPROACH

The NL program was written in LISP and run on a Symbolics computer. The intended users of this NL program would be interacting with an ORACLE database. ORACLE and this database fill were installed on a SUN computer (UNIX operating system). The software developed and described in this report uses the "C" programming language and UNIX operating system calls. A record of the SQL transactions, and corresponding database output is recorded in UNIX files for later review. This provides a valuable debugging tool because when database output is apparently in error, the origin of the error can be isolated to reside in either the database request generating software, or in the database itself.

RESULTS

The objectives were accomplished by using only a couple hundred lines of "C" and "UNIX" software code. As is, this software can interface ORACLE to any program which needs to perform ORACLE accesses. With minor modifications, this program can interface programs with other databases – SQL or otherwise. This program also uses very little CPU time.

RECOMMENDATIONS

The software approach presented in this report provides a simple, robust way to interface a database to another computer program. It is an excellent approach for a development environment where transaction records are needed, and several second delays in data exchange are acceptable. The simplicity of the code accommodates maintenance and flexibility to work with different application programs and database managers. Also, it achieves robustness via simplicity which makes it candidate software for a deliverable system, again where a several second delay is acceptable.

CONTENTS

INTRODUCTION

PROBLEM

A Natural Language (NL) interface which will be connected to a very large expert system is under construction. This expert system is, in turn, connected to a large database. The NL interface, called IRUS, accepts English input and part of its output is a database qucry command. IRUS is developed in isolation from the expert system. To validate the parsing and mapping of the English input to a database query, a direct link between IRUS and the database is needed.

PURPOSE

This report documents the installation of an ORACLE database and its fill with the Fleet Command Center Battle Management Program–Integrated Database (FCCBMP–IDB). Also, this report documents two short "C" and a UNIX shell program to control the input to and output from ORACLE, and the format of the FCCBMP–IDB used by IRUS.

APPROACH

OVERVIEW

IRUS parses English input and if the user is requesting database information, a Structured Query Language (SQL) command is produced. This command is written to a file. IRUS then remotely copies this file (called a target file) over an ethernet to a UNIX computer which is running ORACLE and the software described in this report. A "C" program called ORACLE_front_end polls for the target file and, when it is found, a command is sent to ORACLE to cause it to process the commands in the file. ORACLE processes the SQL commands in the target file and outputs a response. A "C" program called ORACLE_back_end captures the response and writes it to a response file. IRUS is looking for this response file and, when it is produced, its contents are displayed to the user.

The remainder of the approach section describes the five main steps for the UNIX machine: (1) Install ORACLE; (2) Fill ORACLE and database administration functions; (3) The UNIX shell environment and the runme program; (4) ORACLE_front_end; (5) ORACLE_back_end.

INSTALL ORACLE

This section describes the steps taken to install ORACLE version 5.1.17.3 dated August 25, 1987.

- Create an oracle directory with read–write–execute permission for all authorized users.

- Create an oracle login account in the /etc/passwd file.
- Login and go to oracle's home directory.
- Extract ORACLE from the distribution tape via the UNIX command

  tar xvf /dev/rst0 (/dev/rst0 is the cartridge tape drive on the SUN workstation used in this project.
- When logged in as super user in oracle's home directory, run the program rootins.
- Login as oracle and go to oracle's home directory.
- Run the program cinstall.env which completes in a few seconds.
- Run the program dbains. When it prompts for the ORACLE_SID input a single letter. This is the designator which will also be a shell variable. It is also how you designate which instantiation of ORACLE you wish to run. In the instantiation described here, the ORACLE_SID was set to "S". When dbains is done, ORACLE will be installed and ready for use. For more detail consult the ORACLE installation manual (see References).

FILL ORACLE AND DATABASE ADMINISTRATION FUNCTIONS

The fill used in this application came from the FCCBMP–IDB, which is a very large database. To accommodate it, a file for data storage had to be configured and installed. Disk space permitting, a good place to put it is in the ~ oracle/dbs directory (~ oracle means the oracle home directory). For ORACLE efficiency it is preferable to have a few large files rather than many small files. Also, at fill time no table can be spread among files, so large files are required if there are large tables in the fill.

Files are created with the ccf program. The syntax for the command is: ccf filename blocksize. Since the FCCBMP–IDB is so large, and extra space is needed at run time, I used a blocksize of 204800 which is about 104 Mbytes of disk space.

Start ORACLE by typing sqlplus. A freshly installed ORACLE provides one user ID "system" with the password "manager". For security and safety of your database, the system password should be changed immediately. Authorization for use of the database is given by entering the command

```
grant resource,connect,dba to username identified by password;
```

A username of irus was created and granted database administration (dba) privileges. The user can now quit and rerun sqlplus, but this time enter irus as the user id. ORACLE divides its content into partitions. One of these is called the "system" partition. The file created with the ccf command needs to be loaded into this partition. This was done with the ORACLE command: alter partition system add file 'filename';

The copy of the FCCBMP–IDB used in this project came from a VAX computer running ORACLE. ORACLE provides a utility to export its content to make copies on operating system files or tape. The first attempt to transfer the data was unsuccessful. I attempted to export the IDB to files on the VAX, then invoke a EUNIS shell and use the tar program to copy the files onto nine-track tapes. A nine-track tape was mounted on a SUN tape drive and tarred onto the SUN disk. The files were then ftped (file transfer program) to the SUN running ORACLE. The attempt to import these files into the SUN ORACLE failed. Import utility encountered unexpected EOF (end of file) markers in the files.

A successful import of the IDB was achieved using the following steps:

(1) While on the VAX, load the tape in the drive and mount, using the command

```
mount tape/foreign/block=4096
```

Then run the export program, using a block size of 4096, and when it asks for the export filename, enter "tape:" (where tape is the name of the tape drive such as MUA0). This will export the tables directly to tape. If the SUN running ORACLE does not have a half-inch nine-track tape drive mounted on it, only use the first part of each nine-track tape. The reason for this is presented in step 3 below.

(2) Take the nine-track tape and mount it on the tape drive on the SUN. If ORACLE is on a different SUN than this tape drive, then do step 3 below—otherwise skip to step 4.

(3) To transfer the information from the nine-track tape to a quarter-inch cartridge tape, mount one of each tape on their respective drives. Assuming: you are logged into the SUN with the cartridge tape, that its tape drive is /dev/rst0, that the remote SUN (with the nine-track tape drive) is called remote_sun, and the nine-track device is /dev/rmt8; then use the following UNIX command:

```
rsh remote_sun dd if=/dev/rmt8 ibs=10240|dd of=/dev/rst0 obs=10240
```

Since the nine-track tape has a larger capacity than the cartridge tape, any information on the nine-track tape which exceeds the length of the cartridge tape will be lost in the transfer to the smaller capacity tape.

(4) Import the contents of the tape directly into ORACLE. Do not transfer the tape contents to UNIX files first. The import program is named "imp" and is in ~oracle/bin. When imp prompts for the name of the import file, input /dev/rst0 (or whatever device the tape drive is on).

When IRUS runs it makes tables in ORACLE which must be removed periodically. IRUS keeps track of the tables it is making. The IRUS user can initiate a drop of all the tables.

This task can also be done on the SUN by using the following steps:

(1) Start a shell script (following the prompt %, type: script). At this point everything printed to the screen will be copied to a file called typescript.

(2) Step (2) can be done using either of two methods. Method A: Run the program exp (for export). Exp will ask for an output filename. You're only interested in what's being captured in the typescript file, so you can use a junk filename. When it prompts for "entire database", "table", or "user"; enter user. When it asks to export the rows, say no. Method B: Start ORACLE using the sqlplus command. Login in as IRUS. Then type "select * from tab ;". Quit out of sqlplus.

(3) Exit the shell script (hold down the control key and type a lower case "d").

(4) The temporary files created by IRUS contain the string "TMIGHTYFRESH" or "TCAT" or "TFRESH" or some variation thereof. Edit the typescript file so it contains the SQL commands: drop table "tablename" ; where tablename is the actual table name containing the above mentioned strings.

(5) Start ORACLE and give the command: start typescript. The drop table commands which you have just made and stored in the file typescript will then be executed.

THE UNIX SHELL ENVIRONMENT AND THE RUNME PROGRAM

For ORACLE to run correctly, the shell needs to be configured. It is configured in the .login file, and in a file called from .login, the /usr/local/coraenv file. The configuration is performed when the IRUS user starts the top level program called "runme".

The program "runme" (Appendix A) is an executable UNIX shell script. It does four things:

(1) The .login file (Appendix B) is run. The .login file defines a shell variable ORACLE_SID with the value "S". This is relevant because ORACLE was installed with the ORACLE_SID = "S". Lastly, the .login file executes the coraenv file (Appendix C) which further configures the shell variables.

(2) Old target files from previous IRUS runs are removed. The files which IRUS produces are incrementally named targ1.ora, targ2.ora et cetera.

(3) Old response files from previous runs are removed. As with target files, they are incrementally named resp1.ora, resp2.ora et cetera. One response file is created for each target file.

(4) The three programs—ORACLE–front_end, sqlplus, and ORACLE_back_end—are started. The output of ORACLE_front_end is piped (via a UNIX pipe) to the standard input for sqlplus. Sqlplus is a program which invokes ORACLE. Its standard input comes from ORACLE_front_end, and its standard output is piped to ORACLE_back_end.

If runme is unable to start sqlplus, then ORACLE likely needs to be reset. The program runme–reset (Appendix D) is a shell script which clears all users off ORACLE and then does a warmstart. Following this, the user can re-execute the shell "runme" as indicated in the help file (Appendix E).

ORACLE_FRONT_END

ORACLE_front_end (Appendix F) polls once a second for a target file. When one is found, it checks to see if the contents are a valid sql command. If so, the "start filename" command is sent to ORACLE. Polling for the next target file begins. This program is an infinite loop. It stops or exits in response to starting or exiting the running program.

ORACLE_front_end is compiled using the command

make -f front_makefile (Appendix G.1).

ORACLE_BACK_END

ORACLE_back_end (Appendix H) reads the output of ORACLE. The response to each sql command input to ORACLE causes a response output. Each response is written to a response file.

When ORACLE is ready for a command, it gives an "SQL>" prompt. The back_end program reads output from ORACLE until this prompt is read. All the output from ORACLE in between "SQL>" prompts is put in a response file. The back_end reads ORACLE output character by character and buffers the last four characters in memory. If these last four characters are not "SQL>", then the 4th character is written to a buffer file. When the last 4 characters are "SQL>" this buffer file is closed, moved to the next response file, the buffer file is removed, a new buffer file is opened (with the same name), and the "SQL>" buffered in memory is flushed. ORACLE_back_end then resumes looking for output from ORACLE.

ORACLE_back_end is compiled using the command: make –f back_makefile (Appendix G.2).

DISCUSSION

The implementation used in this application was very simple and modular. This produced flexible, robust, quick to implement, and easy to debug software. A trade-off is that up to about two seconds may be lost by exchanging information via file transfers rather than direct sockets between the SUN database program and IRUS. Relative to the time needed for IRUS to parse its English input and generate an sql command, a potential additional delay of up to a few seconds seemed unimportant. In addition to the ease of programming, the strength of the file transfer approach was that; a record of the sql commands produced, and the corresponding ORACLE output could be compared. This facilitates debugging the IRUS software by ensuring that ORACLE's output matches the user's request.

Appendix I shows the format of the data base used. IRUS needs to know the format of the database to identify where the data resides which the user requests. The sql commands produced by IRUS show how IRUS mapped the user's requests into a database request. Having this on line in a file is convenient for debugging.

RECOMMENDATIONS

The software approach presented in this report provides a simple, robust way to interface a database to another computer program. It is an excellent approach for a development environment where transaction records are needed, and several second delays in data exchange are acceptable. The simplicity of the code accommodates maintenance and flexibility to work with different application programs and database managers. In addition, it achieves robustness via simplicity which makes it candidate software for a deliverable system, again where a several second delay is acceptable.

REFERENCES

"ORACLE for SUN Workstation/SUN UNIX System Release Bulletin," ORACLE Corporation, Belmont, California 94002, Part No. 416-V5.1 August 4, 1987.

APPENDIX A

```
##########################################################################

                              ## runme ##

##########################################################################

              ## Top level Program to Handle the SUN (UNIX) ##
                 ## End of the IRUS to ORACLE exchanges ##

#  This shell program invokes three programs and controls the exchange
#  of data amongst them via UNIX pipes.

#  Prior to the above, the .login file is run which contains commands
#  to configure the shell environment for proper execution of ORACLE.

#  The old target and response files are removed.

#  Any ORACLE core dumps from previous runs are removed. These core
#  dumps are in the same directory as the executable sqlplus.

#  The last line starts the three main programs. This line starts
#  the front and back end processors. The pipe directs the standard output
#  of the process on the left into the standard input of the process to
#  its right. The pipe symbol is: |

#  ORACLE_front_end looks for a target file once each second. When one
#  is found, ORACLE is told to process that file (via the "start filename"
#  command), and polling for the next target file is begun.

#  The output of ORACLE is captured via a UNIX pipe by the ORACLE_back_end
#  program. This data is buffered in the file back_scr, and when all of
#  the ORACLE output is received, the back_scr file is moved to a response
#  file. The ORACLE_back_end program is then poised to capture the
#  next output of ORACLE and write it to the next response file. Response
#  files are sequentially named resp1.ora, resp2.ora ... and correspond to the
#  target files named targ1.ora, targ2.ora ... respectively.

source .login
rm targ*.ora &
rm resp*.ora &
rm ~oracle/dbs/core* &
echo 'Done initializing'
ORACLE_front_end | sqlplus | ORACLE_back_end
```

APPENDIX B

```
################################################################################
                              ## .login file ##
################################################################################

set term=sun
set history=30
alias RUNME runme

#   Set up enviroment for access to the ORACLE database.
#   This assumes that the local bin directory is "/usr/local"

set path=(. /usr/ucb /usr/lib /etc /usr/etc /bin /usr/bin /usr/local)
set path=($path ~irus ~irus/oracle ~irus/oracle/bin )
set noglob; eval 'tset -Q -s'; unset noglob

#   Each ORACLE instanciation on a machine should be setup with a unique
#   ID. This way a user has a handle to specify which instanciation he wants
#   to execute.

setenv ORACLE_SID "S"

#   Configures the shell.

source /usr/local/coraenv
```

APPENDIX C

```
################################################################################
                              ## coraenv file ##
################################################################################
#
#   $Header: coraenv,v 1.3 86/11/25 13:16:15 choplin Exp $ coraenv Copyr (c) 1986
Oracle
#
#
#   Copyright 1984, 1985, 1986 Oracle Corporation
#
#   This routine is used to condition a C shell user's environment
#   for access to an ORACLE database. It should be called from
#   the user's ".login" file. The database SID, if any, should
#   be set into the environment prior to executing this routine.
#
#   usage: setenv ORACLE_SID=SID
#   source coraenv
#
########################################
#
#   Set minimum environment variables
#
if (${?ORACLE_SID} = = '0') then
    setenv ORACLE_SID ""
endif
setenv ORACLE_HOME 'dbhome $ORACLE_SID'
if (${?ORACLE_REL} = = '0' ) then
    setenv ORACLE_REL 'cat ${ORACLE_HOME}/dbs/ORACLE_REL'
else if (${#ORACLE_REL} = = '0') then
    setenv ORACLE_REL 'cat ${ORACLE_HOME}/dbs/ORACLE_REL'
endif
#
#   Adjust path accordingly
#
#   This ensures that only one occurrance of "$ORACLE_HOME/bin" is in path
set STRIP='echo ${ORACLE_HOME}/bin | sed "s./.\\\/.g"'
set path=(${ORACLE_HOME}/bin 'echo $path | sed "/${STRIP}/s///g"')
unset STRIP
#
# Install any "custom" code here
#
```

APPENDIX D

```
########################################################################
                           ## runme-reset ##
########################################################################

# This program is to be run when the runme program fails due to ORACLE
# not starting correctly.

# The first command clears ORACLE. Any current users are knocked off
# without warning. The second command warmstarts ORACLE.
# After running this program, runme can be rerun.

ior clear
ior warmstart
```

APPENDIX E

```
############################################################################

                                 ## help ##

############################################################################


#  This file prints a help message on the screen to aid the user in
#  starting the ORACLE package.

echo ''
echo 'To start type: runme '
echo 'To exit type: control-z <cr>'
echo ' and then type: kill %'
echo ''
echo 'If you do not get the message "ORACLE started successfully" '
echo '  when running runme, exit runme as described above.'
echo '  Start the program runme-reset, which will reset ORACLE '
echo ''
echo 'Restart runme '
echo ''
```

APPENDIX F

```
                    /*  ORACLE_front_end  */

#include <stdio.h>

open_read_close(filename)
    char filename[128];
    {
    int fd, fd1;
    FILE *fp;
    char cat_command[128];          /* Will store the cat command */
    char buffer[2048];              /* Buffer for a 2048 byte block */
    char small_buffer[6];           /* Buffer for a 6 byte block */
    int bytes_read;

    /* Open filename and read the last character */
    fd1 = open(filename, 0);
    if (fd1 == -1)
        fprintf( stderr, "\n%s%s%s\n", "Error:  Unable to open ",
            filename, " in front_end.");

    else {
        bytes_read = read(fd1, buffer, 2047);
        lseek(fd1, -5L, 1);         /* Start lseek five characters from the end*/
        read(fd1, small_buffer, 6);
        close(fd1);                 /* Close the file. */

    };

    /*  If the last character is not a semi-colon (which is how SQL
        commands end) then close the file, and wait for 2 seconds to see
        if any more will be added to the file. If after the 2 seconds
        the file still does not end in a semi-colon, then assume that
        the file transfer was messed up, and exit */

    if ( small_buffer[0] != ';' && small_buffer[1] != ';' &&
        small_buffer[2] != ';' && small_buffer[3] != ';' &&
        small_buffer[4] != ';' && small_buffer[5] != ';') {
        system("sleep 2"); /* Sleep for 2 seconds. */
        fd1 = open(filename, 0);    /* Open the file a second time. */
        if ( fd1 == -1)
            fprintf(                stderr, "\n%s%s%s\n", "Error: Unable to open ",
                filename, " in front_end.");

        else {
            bytes_read = read(fd1, buffer, 2047); /* Find it's length in bytes */
            lseek(fd1, -5L, 1);     /* Go a few places before the ";". */
            read(fd1, small_buffer, 6);     /* Hopefully read the semi-colon */
            close(fd1);

        };
        }; /* End if small_buffer != ; */
        /* If it's still not a semi-colon, print an error message and give up. */
            if (small_buffer[0] != ';' && small_buffer[1] != ';' &&
            small_buffer[2] != ';' && small_buffer[3] != ';' &&
            small_buffer[4] != ';' && small_buffer[5] != ';') {
            fprintf( stderr, "\n%s%s\n%s%s%s\n", "ERROR: The file ", filename,
            " ended with ", small_buffer,
            " which should have contained a semi-colon.");

        return;

    };

    /* If the file ended in a semi-colon, this part of the code is reached.
        The "start filename" command is sent to ORACLE which causes it to
        to process the contents of filename */
    /* The next seven lines are equivalent to : printf("start %s\n", filename)
        but for some strange reason, the UNIX pipe doesn't catch the
        printf output, but is does catch the output from the "cat" command */
    fp= fopen("front_start", "w");
    if( fp != NULL) {
        fprintf( fp, "start %s\n", filename);
        fclose( fp);
        sprintf( cat_command, "cat front_start");
        system(cat_command);
        fprintf( stderr, "\nStarting %s.\n", filename);
    }
    else fprintf(stderr, "\nERROR: Unable to access front_start.\n");
}; /* end open_read_close */

    main( ) /* ORACLE_ front_end */
    {
    int counter = 1;
    char filename_buffer[128];        /* A buffer to hold a filename string */
    char ls_command[128];             /* Will store the ls command */
    char incoming_filename[128];      /* The next file to retrieve from IRUS */
    FILE *fp, *fopen();

    system("sleep 2");
    sprintf(ls_command, "cat %s", "ORACLE_login_script"); /* This file
        contains your ORACLE login name and password */
    system(ls_command);
    sprintf(incoming_filename, "targ%d%s", counter, ".ora");
    while(1) {
        sprintf(ls_command, "ls %s > front_scr", incoming_filename);
        system(ls_command);
        fp = fopen("front_scr", "r");
        fscanf(fp, "%s", filename_buffer);
        fclose(fp);
        if (filename_buffer[0] = = 't') { /* If a file was found */
            open_read_close(incoming_filename);
            /* Set things up to start looking for the next incoming file */
            sprintf(filename_buffer, "%s", " ");
            counter ++;
            sprintf(incoming_filename, "targ%d%s", counter, ".ora");
        }
        else system("sleep 1");              /* No file found, so pause for a second
                                             before looking again. */

    }; /* End while(1) */

}; /* End main ORACLE_front_end */
```

APPENDIX G.1

```
##############################################################################

                ## Makefile for ORACLE_front_end ##

##############################################################################

ORACLE_front_end: ORACLE_front_end.o
    cc -g ORACLE_front_end.o -o ORACLE_front_end
ORACLE_front_end.o:
    cc -g -c ORACLE_front_end.c
```

APPENDIX G.2

```
##############################################################################

                ## Makefile for ORACLE_back_end ##

##############################################################################

ORACLE_back_end: ORACLE_back_end.o
    cc -g ORACLE_back_end.o -o ORACLE_back_end
ORACLE_back_end.o:
    cc -g -c ORACLE_back_end.c
```

APPENDIX H

```
                              /* ORACLE_back_end */

#include <stdio.h>

main( )
    {
    int counter = 0;              /* Counter to produce output names */
    int header;                   /* Used in stripping off the ORACLE prompts */
    int look;                     /* Continue looking for ORACLE output */
    char c, c1, c2, c3, c4;       /* One character of output from ORACLE */
    char line[1024];              /* Buffers one line from the ORACLE output */
    char response_file[128];      /* File for output */
    char cp_command[128];         /* Buffer to hold the system copy command */
    FILE *fpin, *fpout, *fopen();

    /* Read in and throw away initial ORACLE login stuff */
    sprintf( line, "  ");
    look = 1;
    while(look) {
        scanf("%s", line);
        if(line[0] == 'S') {
            if(line[1] == 'Q') {
                if(line[2] == 'L') {
                    if(line[3] == '>') {
                        look = 0;

    };};};};};

    fprintf( stderr, "\n   ORACLE started successfully\n");
        while(1) {                    /* Always true (infinite loop) */
            unlink("back_scr");
            fpout = fopen("./back_scr", "w"); /* Open the output file */
            if ( fpout == NULL) {
                fprintf( stderr, "\nERROR:  Unable to open back_scr in back_end.\n");
                system("sleep 1");

        };

         /* Create the next response file name */
        counter ++;
        sprintf( response_file, "resp%d%s", counter, ".ora");
        sprintf( line, "    ");

        look = 1;
        c4 = '\0';
        c3 = '\0';
        c2 = '\0';
        c1 = '\0';

        while(look) {
                c = '\0';
                c = getc(stdin);
                if (c != '\0') { /* getc found something */
                c4 = c3;
                c3 = c2;
                c2 = c1;
                c1 = c;
                if ((c4 == 'S') && (c3 == 'Q') && (c2 == 'L') && (c1 == '>')) {
                look = 0;
                c4 = '\0'; c3 = '\0'; c2 = '\0'; c1 = '\0';

            }
            else {
                if (fpout != NULL) fprintf(fpout, "%c", c4);
            };
        };
    };

    if (fpout != NULL) fclose(fpout);
    sprintf( cp_command, "cp back_scr %s", response_file);
    system(cp_command);
    fprintf( stderr, "\nCompleted making %s.\n", response_file);
    };

}; /* End main ORACLE_back_end */
```